Mathematics and computer science: The interplay

C. E. Veni Madhavan

Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560 012, India

Mathematics has been an important intellectual pre-occupation of man for a long time. Computer science as a formal discipline is about seven decades young. However, one thing in common between *all* users and producers of mathematical thought is the almost involuntary use of *computing*. In this article, we bring to fore the many close connections and parallels between the two sciences of mathematics and computing. We show that, unlike in the other branches of human inquiry where mathematics is merely utilized or applied, computer science also returns additional value to mathematics by introducing certain new computational paradigms and methodologies and also by posing new foundational questions. We emphasize the strong interplay and interactions by looking at some exciting contemporary results from number theory and combinatorial mathematics and algorithms of computer science.

Nature of the sciences

MATHEMATICS has been an important intellectual pre-occupation of man for a long time. It is said that the *mathematics of the common man* or the *mathematics of the millions* does not possess the esoteric abstractions and expressions of the *mathematics of the professional mathematician* or the *millennium mathematics problems*. However, one thing in common between *all* users and producers of mathematical thought is the almost involuntary use of *computing*. The nature, intensity and extent of the actual computing may differ quite widely among these different forms of engagements with mathematics and computing. The basic processes of empirical verification, development of abstractions from concrete instances inherent in mathematical thinking has much in common with algorithmic or computational thinking.

Mathematics is as old as humanity, while computer science is a young discipline, about seven decades old. The strong interplay between mathematics and computer science is at its peak today.

Much has been written by philosophers on the nature of human inquiry. Yet, it is quite difficult to define a term such as mathematics in a comprehensive and succinct manner. A whimsical, circular definition states that mathematics is what is practiced by mathematicians. It is not possible to attempt even such an indicative definition of the discipline of computer science. Computer science is perceived as an inter-disciplinary field with clear tracks of contributions from electrical and electronic engineering way of thinking and a clear track of contributions from the mathematical way of thinking. Further, these two tracks ramify into many different sub-tracks and rejoin after a few complex interactions. At a broad level computer science thus straddles simultaneously scientific, engineering and technological principles.

The term computing science seems to refer to the act of computing whereas the term computer science seems to refer to the principles of architecting and engineering a computer. Indeed, it is true as seen from the previous sentence, computer science sometimes refers to all of these and more. We will use somewhat synonymously the terms, computer science and computing science. The other terms computational science, science of computing, scientific computing, mathematics of computation, computational mathematics all have slightly different specialized meanings and occasionally allow some overlap. We will be conscious that we sometimes broaden our boundaries. The purpose of this article is not to delineate and distinguish clear demarcations. On the other hand the intent is to bring out the beautiful connections, similarities and parallels between the two disciplines of *mathematics* and *computer science*.

In our work we consider only the facet of theoretical underpinnings of computer science leaning on the science of algorithms. We do not consider the meta-physical, cognitive connections and hence do not make forays into topics of formal logics, computational linguistics, artificial intelligence, machine learning and other related fields. Also we do not discuss the exciting new possibilities of computing machines based on the quantum mechanical or biological processes, that may render as irrelevant some of the deep theoretical and practical issues we study here.

The early formalisms in computer science have made clear the connections among the three entities of machines, languages and computation. They also established these connections by formalisms based on various forms of automata, the corresponding formal language classes and the generative mechanisms of grammars. Machines are synonymous with algorithms in a formal sense. From these theoretical underpinnings arose the formal specifications of algorithms and programs. The evolution of the theoretical foundations of computing science took place along these two clear tracks – algorithms and programs.

e-mail: cevm@csa.iisc.ernet.in

The older form of the word algorithm is *algorism*. This term refers to the process of doing arithmetic using Arabic numerals. Mathematical historians trace the etymology of the word *algorithm* to the word algorism. The term comes from the name of a Persian author, Abu Musa al-Khowarizmi (c. 825 AD)[1], who wrote a celebrated book, *Kitab al jabr w'al-muqabala* (Rules of Restoration and Reduction). The word algebra stems from the title of this book. The ancient town of Khowarizm is the modern town of Khiva in Russia.

An *algorithm* is a compact, precise description of the steps to solve a problem. It is a finite, definite, effective procedure taking an input and producing an output. A *program* is a precise set of statements expressed using the syntactic, semantic and linguistic constructs of a programming language, that embodies the steps of an algorithm. The remarkable evolution and applications of computers is due to the immense range of developments in the science of algorithms and programs. The two main issues in the study of algorithms and programs are *correctness* and *computational complexity*. Many theoretical and pragmatic ideas along these directions have led to remarkable advances.

We discuss the notion of proofs in mathematics and the role of computers and computing in this context. We discuss the issues of correctness and computational complexity in the context of design and analysis of algorithms. It is here that many fascinating connections between mathematics and computing science appear in many surprising ways. These connections have led to very exciting developments in both fields of inquiry.

We do not discuss in this paper, the formal methods for reasoning about programs and their intrinsic relationship with certain abstract structures of mathematical logic and algebraic structures. We merely cite the excellent works and treatises of the pioneers Dijkstra[2] on the science of programming, of Wirth on the notion of data structures which together with algorithms leads to the realization of programs and of Manna[4] on the logics used to reason about the correctness of programs.

Algorithms and computational complexity

A lot has been said and written on the topic of design and analysis of algorithms. There are a large number of great text books, advanced monographs and high quality specialized journals and conferences in this area. They cover a range of issues related to design strategies (such as greedy, divide and conquer, branch and bound, binary doubling, etc.), implementation (choice of data structures to store and process information), and analysis (model of computation, asymptotics, lower bounds, structures in complexity, etc.). We focus only on the larger macro level features and their relationships with mathematics.

A beautiful summary of the common features shared by algorithmic thinking and mathematical thinking is made by Knuth[5]. The two forms of thinking are characterized by features such as formula manipulation, representation of reality, reduction to simpler problems, abstract reasoning. The notable differences in features are the way of handling the uncountable continuum in mathematics and the way of handling the notion of size of proof (computational complexity) in computer science.

The algorithm of Euclid (*Elements, Book 7*) for calculating the greatest common divisor (gcd) of two integers is a fine example of the modern notion of algorithm. It displays all the properties and features related to establishing the correctness and computational complexity. One measures the computational complexity of an algorithm by expressing the number of basic steps taken by the algorithm to solve a problem, in terms of the sizes of the input (and the sizes of all the intermediate results). In general, the maximum number of steps taken over all possible inputs over asymptotically large-sized inputs is used as an upper bound and serves to characterize the behaviour of the algorithm. This measure, expressed as a function of the input size is known as the *worst case asymptotic* (*time*) *complexity* of the algorithm.

The size of an element is usually the number of binary bits needed to encode the element. Let the sizes of the two input integers x, y be n bits. The Euclidean gcd algorithm to obtain the gcd(x, y) requires number of steps (or equivalently time) proportional to $O(n^3)$. Thus the computational complexity of the Euclidean gcd algorithm is said to be cubic in input size. An algorithm whose complexity is polynomial in the input size is considered *good*. A problem that admits a polynomial time algorithm is said to be *tractable* or *easy*.

On the other hand there are a large number of problems, that do not seem to be *easy* (i.e. they do not seem to be solvable by means of a polynomial time algorithm). Consider the mathematical structure *graph*. Let $G = (V, E)$ be a graph, where V is a set of n *vertices*, and E is a collection of *edges* (a subset of the set of $n(n-1)/2$ pairs of vertices). The size of the input in the case of a graph is proportional to the number of vertices and edges and is bounded by $O(n^2)$. For example, the problem of determining whether a *graph* G has a *Hamiltonian cycle* (a cyclic traversal of the edges of the graph visiting every vertex exactly once), does not seem to be *easy* in the above technical sense. Clearly, by listing all possible $n!$ permutations of the vertices and checking whether the graph can be traversed along the edges as per the vertex permutation, one can solve the problem. It is obvious that this algorithm, adopts a *brute-force* approach of exhaustive enumeration of all possible solutions. The computational complexity is *exponential* in the number of vertices since the function $n!$ grows approximately as $O(n^{n+1/2}\exp^{-n})$.

Proofs in mathematics and computer science

In the previous section, we discussed how we assess the computational complexity of an algorithm. There is the other aspect of *checking* the answer produced. How do we know that the answer produced by an algorithm or the realization of the algorithm is *correct*? Is there a simple mechanism, a formula, an expression into which we can plug in the answer and check the answer? For a simple example, consider the problem of subtraction of integer b from integer a. We immediately recognize that we could add the result of subtraction to b and check for match with a. Now, what is the complexity of carrying out this verification? In this case, addition and subtraction are both *linear* in the length of the operands a and b and hence the verification can be carried out in *polynomial* time and hence we say that the verification is *easy*. Here we verified the correctness of an algorithm by *using* another algorithm. Of course we assumed that the other algorithm produced correct answers and that the result of that can be verified easily.

The answers to problems that have polynomial time algorithms for producing the answers seem to be verifiable in polynomial time (see the subtraction problem) by running another complementary algorithm. Another pair of simple, complementary algorithms are the multiplication and division algorithms with quadratic running time. This situation prevails in a non-obvious way in many situations. The non-obvious nature comes from the role of the *mathematical theorems* that characterize the situation. For example, in the case of the greatest common divisor of integers, there is a theorem due to Bezout, that states that if $\gcd(x, y) = g$, then there exist two integers u, v such that $ux + vy = g$. The Euclidean algorithm can be *extended* to determine the constants u, v besides g, given x, y. While the Bezout relation can be used to check if the gcd g is correct, assuming the quantities u, v are correct. The catch is that u, v are also obtained by the same algorithm. However, there is another way. From the fundamental theorem of arithmetic and the consequent unique factorization theorem the $\gcd(x, y)$ is immediate from the prime factorizations of x and y. Now we have a different catch. Factorization of an integer is not known to be polynomial time solvable. Much of our current research is centered around this problem.

In the next section, we discuss the formal structures that cope with these kinds of situations. The main message here is that *independent characterizations* enable the determination and verification of an *answer*. It is not at all clear whether two or more independent characterizations would be available for a problem. It is not true that the different characterizations will lead to the development of polynomial time algorithms to produce the answer. Such situations are dealt with in problems of optimization, by means of a variety of duality, reciprocity and min-max theorems. We discuss these in a subsequent section. Although the initial formal notions of algorithm and computational complexity figured most prominently in the computer science literature during the1970s, the notion made its first appearance in a classic paper by Edmonds[6] on maximum matchings in graphs.

On the other hand, many problems have the peculiar characteristic that the answers could be verified in polynomial time; but they do not seem to admit polynomial time algorithms to find the answer (see the Hamiltonian cycle problem). It is clear that the *Yes* answer to the question of Hamiltonicity of a graph can be verified in linear time once a Hamiltonian cycle is given. This situation prevails in the case of many problems of different nature (decision, search, optimization, etc.), arising in different domains (sets, integers, graphs, etc.) and from many different application areas (scheduling, information retrieval, operations research, etc.).

Computer scientists realized quite early, that there is a philosophic difference between the computational complexity of an algorithm used to solve a problem and the computational complexity of verification of the answer. They also put on a firm formal footing these notions of algorithmic production of a solution and that of verification. They also set up, in the spirit of mathematical abstraction, equivalence classes of problems categorized according to their computational complexity. The equivalence class of problems whose answers could be verified in polynomial time was called **NP** and the former class **P**. Clearly **P** is contained in **NP**. An important result was the postulate of an equivalence class of *hard* problems within **NP** called the **NP-complete** class. An intriguing situation in the realm of computational complexity is that there does not seem to be easily verifiable *succinct certificates* to the **No** answers to decision versions of many problems. For example, it is not at all known how one can provide a short polynomial time verifiable proof to the answer that a given graph does *not* have a Hamiltonian cycle.

Computer scientists have built many interesting sub-structures within **NP**. The fundamental meta-scientific question is whether **P** is equal to **NP**. This question has engaged computer scientists for the last three decades, in developing formalisms and a lot of technical hair-splitting. We will not go into further details of this fascinating question from the foundations of theoretical computer science. We refer the reader to the classic book on this subject by Garey and Johnson[7]. We just make a few comments on the current perspectives on this subject which borders on meta mathematics.

I would borrow the quaint little phrase *Meta-magical Themas* used by Douglas Hofstadter as the title for his column in the journal *Scientific American*, to describe these questions. Interestingly, this phrase was coined by Hofstadter, a computer scientist, as an anagram of the title *Mathematical Games* used by Martin Gardner for many years as the title of his column in *Scientific American*.

The Clay Mathematics Institute of Massachusetts, USA, named seven Prize problems, each carrying a cash award of one million dollars, in May 2000. These include the celebrated Riemann hypothesis, and a few other outstanding conjectures of twentieth century mathematics. These problems are toasted to be on the same class of problems stated by D. Hilbert in the year 1900. The eminent French mathematician, J. P. Serre, commenting on these problems, said that the choice by the Clay Institute was very apt. He hoped that the **P** = **NP** question would not turn out to be *undecidable*. This question perhaps, is a meta question and is on the same footing as certain axiomatic questions in the foundations of set theory upon which the entire edifice of modern mathematics rests.

Thus the notion of quickly verifiable *succinct proof* or a short certificate to the correctness of the solution determined by an algorithm, is a cornerstone contribution of computer science to mathematical thought. In the following sections we look at the notion proofs in mathematics and some close connections with computational thought.

Many proofs and many algorithms

In mathematics the notion of proof is a central ingredient of all results. No mathematical statement is made without a conscious attempt to establish or prove the statement. Mathematicians are taught and trained in the course of their development from a student to a professional practitioner, the science of proving statements. Mathematical statements consist of a set of axioms, hypotheses, antecedents and consequents. The statements are established by a sequence of logical consequences from the hypotheses and antecedents to the consequents by a chain of logical deductions. Of course, mathematics is not to be construed as a mere set of dry, mechanical deductions. If that were so, such a human endeavour would not have survived. There is a certain innate beauty in many mathematical statements and theorems and very often in the many twists and turns of the proofs. One wonders at times if this innate quality of mathematics is akin to the surrealistic patterns of human inquiry in fields such as the fine arts, literature, music, painting and sculpture. Indeed, the famous mathematician Littlewood once described eloquently certain results of Srinivasa Ramanujan as evoking the sense of wonder produced by looking at a certain ecclesiastical architectural masterpiece.

Many mathematicians and philosophers have pondered over this *innate* quality of mathematical thought. This innate quality transcends the physical world. Scientists in many disciplines have come across a similar phenomenon in explaining their science. Mathematical statements and proofs have an existence of their own, as pure figments of imagination. However, there is evident in many of these forms of expressions of thought beautiful insight into the structure of the world of mathematical objects. In general, these objects are used to model the physical world and the statements are about the way these objects interact to produce more complex structures with real world consequences.

In modern times, mathematicians and computer scientists have contributed many significant ideas that have all the features discussed above. In the last few decades computer science has emerged as a mature foundational science, with its own corpus of axioms, statements and body of results. It is here that the science of algorithms, or sometimes referred to as algorithmics (particularly by the French research community), stands out like a beacon proclaiming its status on par with mathematics. The fraternity of mathematicians and computer scientists have come to recognize and acknowledge with mutual respect certain exquisite results from their respective disciplines. Of course, mathematics being a much older science has many more theorems and results of great beauty and significance than the algorithms of computer science. Increasingly, computer science is enriching mathematics in many ways. This phenomenon is most visible in the area of *discrete mathematics*. We look at this phenomenon more closely in the next section.

We discussed above the conceptual parallels between the nature of proofs in mathematics and the nature of algorithms in computer science. We take this further to identify certain other similarities in our paper. We identify two typical characteristics. (i) *many alternative proofs* (ii) *short*, *succinct* or *elegant proofs*. Both characteristics are present in abundance in the sciences of discrete mathematics and algorithms. In fact, these are precisely the reasons that make these two subjects so fascinating. It is tempting to cite and annotate the several examples to illustrate these two features from the two disciplines. I shall restrict to just a couple of illustrations to make the point.

Many proofs

The *law of quadratic reciprocity* of Gauss is lauded as the gem of arithmetic by the mathematical historian, E. T. Bell. This law expresses the quadratic residuosity of a prime integer p modulo another prime q, in terms of the quadratic residuosity of q modulo p in the form of a surprisingly simple expression in terms of the parities of p and q. As a mathematical statement this is indeed a gem. It is beautiful, simple and so natural to think up. What is more, this law occupied the centre stage of number theoretic research during a large part of the 19th century. It is not surprising that Gauss himself discovered eight proofs, and a 152nd proof of the law was published[8] in 1976. The hunt for a generalization of this law, meaning higher order reciprocity connecting solutions of higher power congruences modulo prime integers, occupied the legendary mathematicians Gauss, Kummer, Jacobi, Dirichlet. Finally, Eisenstein obtained a proof of the law in 1865. What fascinates me besides these interesting *many* proofs by *many* mathematicians is the *many* connections to many other *problems* that have led to surprising new developments. Indeed, Gauss, Eisenstein, Kummer, Dirichlet and others were all seeking proofs of higher power reciprocity laws (see Edwards[9] and Lemmermeyer[8]) while attempting to prove Fermat's last theorem. In the course of this quest, Kummer had a big success in proving the Fermat's last theorem for the class of *irregular prime* exponents and paved the way for the development of modern class field theory.

Mathematicians devise ingenious *different* proofs by applying their intuition together with techniques and principles from diverse branches of mathematics in surprising ways.

For example, the different proofs of the law quadratic reciprocity cited above draw upon principles from elementary enumerative combinatorial arguments to sophisticated use of algebraic structures and complex analytic tools. To a specialist mathematician, the hundreds of proofs mentioned above would all fall into a few categories. Nevertheless, these few categories encompass surprisingly different principles. That speaks for what is known as scientific serendipity without which much of the science of mathematics would lose its charm.

The famous *prime number theorem* states that the number of primes less than an integer x is about $x/\log(x)$. This simple statement about the lore of integers has fascinated number theorists for a long time. The proof of this theorem by Hadamard and de la Vallee Poussin is a classic example in the world of mathematics of a less well-known mathematician's name being associated with a great result. Surprisingly, this and many other similar, related results in the fascinating branch of mathematics called number theory[10,11], invoke simultaneously techniques for dealing with the discrete and the continuous. At a deeper level, this is not too surprising. As Kronecker, put it *God made natural numbers*, *rest is the work of man*. A natural mathematical approach is to begin modelling and analysing a finite, discrete world and then entering the continuum by adopting an asymptotic limiting process. I shall not go into further explanations of this metaphor.

I end with the comment about the alternative proof of the prime number theorem by Erdos and Selberg. This proof used combinatorial arguments entirely, and did not use any complex analysis techniques. The authors called their proof an elementary proof. This proof is an illustration of what we term, elementary but not easy. It is elementary in the sense that it does not use advanced techniques available only to the professionally trained mathematician. It is not easy in the sense that the arguments are involved and intricate. However, this proof is *longer* than the analytic proof. We use the adjective longer here, to indicate the degree of difficulty in wading through the verification of several minute details. Mathematicians, like artists, sometimes refer to the quality of a proof by the term *elegance*. We will consider this notion of *mathematical elegance* in the next sections.

Many algorithms

Mathematicians are coming around to the viewpoint that *algorithms* as studied in computer science could be considered very close to the object they study, called *theorems*. The basis for this acceptance has been discussed above. We further add here that in the last two decades, many prestigious fora and awards that recognize significant contemporary mathematical results have welcomed the fresh breath of exciting mathematical results from computer science.

Computer scientists have devised many interesting algorithms for the same problem. In doing this, they have often blended great insights from the structures in the problem domain, the structures inherent in the input data and certain insightful generic algorithmic strategies. I give some sophisticated examples later. Here we mention a few basic problems for which computer scientists have come up with surprisingly different ways of solving them. These are: (i) sorting a set of elements[12], (ii) multiplication of polynomials, integers and matrices[10], (iii) finding shortest paths in graphs[13], (iv) rapid exponentiation in groups[10].

Some of the brilliant, and far reaching ideas from the world of algorithmics, that have enriched the mathematical world come from topics such as polynomial time interior point-based algorithms for the linear optimization problem, equivalence of combinatorial optimization problems in terms of computational complexity; the applicability of lattice basis reduction for designing algorithms for optimization problems; the notions of approximation, randomized algorithms; algorithmic combinatorial geometry; polynomial time primality testing algorithm for integers; modular exponentiation-based public key cryptographic algorithms, syndrome decoding algorithm in error correcting codes.

What I set out to illustrate in this section is the situation of *many* algorithms for solving a problem, like the situation of many proofs for the same theorem in mathematics. These *different* algorithms for the same problem provide a vast repertoire of illustrative pedagogic, research principles and techniques that enliven the science of algorithms. They provide the continual challenges to the scientists to come up with *better*, *elegant*, *efficient and practical* algorithms.

Elegant proofs and elegant algorithms

The legendary twentieth century mathematician Paul Erdos liked to talk about *The Book* in which God maintains the most elegant or perfect proofs. The Hungarian mathematician was known as the most prolific mathematician of all time, comparable to the other prolific mathematician, Euler of an earlier era. Erdos maintained that even if one did not believe in the existence of God, one should believe in the existence of The Book. Aigner and Ziegler[14], made a selection of 30 items in a collection published in 1998. It is an unenviable task to assemble such a collection. A certain amount of mathematical chauvinism is inevitable and so there are possibilities of omission in the selection. However, it is irrefutable that what figures in such a selection is indeed elegant.

I cite a few from this collection.

- For any positive integer n there is a prime between n and $2n$.
- A natural number n is a sum of two squares if and only if every prime factor p of the form $4m + 3$ appears to an even power in n.
- π, π^2, $\exp^r$ for rational r, are all irrational.

- If G is a connected, planar graph, then $n - e + f = 2$. Here n, e, f are the number of vertices, edges and faces in G.
- No more than 12 spheres can simultaneously touch a sphere of the same size.
- There are n^{n-2} different labeled trees on n vertices.
- A partial Latin square of order n with at most $n - 1$ filled cells can be completed to a Latin square of the same order.
- Every planar map can be five coloured.
- For any museum with n walls $n/3$ guards suffice.
- A triangle-free graph on n vertices has at most $n^2/4$ edges (generalizations exist for p-clique-free graphs for all p).

The main intent in giving the list above is to display the simplicity of the statements. Other interesting aspects are that none of the proofs run for more than a few pages; and are accessible to anyone with a good high school level of mathematical training. That is the nature of *succinctness* or *elegance* of mathematical thought.

It would be extremely fascinating to think up *The Book* of algorithms. Computer science is perhaps, nearly ready to offer candidates. I could make the following random selection, without much trepidation.

- The Euclidean algorithm for greatest common divisor.
- The quick-sort algorithm.
- Algorithm to test the planarity of a graph.
- Finding the maximum matching in a graph.
- The randomized algorithm for the roots of a polynomial over a finite field.
- The Fast Fourier transform algorithm.
- The deterministic polynomial time primality testing algorithm.
- Determination of the convex hull of a set of points in 3 dimensions.
- Run length compression of binary strings.
- Set union, search algorithm.

The interplay

In the above sections we discussed the principal features of *theorems* of mathematics and *algorithms* of computer science to bring out the close parallels, connections and also some differences. The intent was to display the nature of thinking: *mathematical* and *algorithmic*. In this section, we develop this idea and show how these two disciplines have enriched each other. Significant developments in the field of combinatorial mathematics have happened in the last few decades alongside *related* developments in computer science. Often questions are posed across the disciplines and the results obtained have a bearing on each other.

In combinatorial mathematics one begins with a *ground set* and then studies certain objects built from the elements of the ground set, called *configurations*. For example, permutations, combinations are simple configurations on discrete sets. More sophisticated configurations are partitions and collections with specific properties. Various ground sets made of elements of algebraic structures (groups, residue classes modulo prime integers, elements of finite fields, vectors in a vector space), geometric structures (points, lines, hyperplanes, regions, polyhedra), arithmetical structures (integers, rationals, Diophantine equations, equations over finite fields) are all of great interest.

In the field of combinatorial mathematics, the questions posed fall into three classes: *existential*, *enumerative* and *constructive*. Existential combinatorics is concerned with questions of proof of existence or non-existence of certain specific types of configurations. They also seek to obtain mathematical characterizations or necessary and sufficiency conditions for the existence.

For example, the famous condition by Kuratowski states that a graph is planar if and only if it does *not* have a subgraph *homeomorphic* to the complete graph on 5 vertices, denoted a K_5, or a complete bipartite graph on 6 vertices, denoted a $K_{3,3}$. The proof of this theorem does not lend itself to an algorithm for checking the condition. Thus the proof is purely existential and not effective. However, the proof can be modified to a version that suits an algorithmic determination of planarity, by actually constructing a planar embedding or reporting impossibility if the graph is *not* planar. Further, the famous planarity testing algorithm of Hopcroft and Tarjan works in *linear time*. It uses a special data structure. Contrast this situation with that of the computer based proof of the *Four Colour Theorem*. We mentioned in the list in the previous section, that there is a nice theorem with a short proof to the fact that every planar graph can be coloured with 5 colours. The proof by Appel and Haken[3] of the 4 Colour Theorem, required a voluminous inspection of various sub-cases, using a computer to eliminate *unavoidable configurations* after formally, reducing the general situation to this large set. Much debate has taken place about the *nature* and hence the acceptability of this kind of proof. As a scientific work, this proof is as rigorous as any in mathematics. The dissatisfaction about this kind of proof stems from the fact that it seems to have to consider *many cases*. The purist likes to see a *short*, *elegant*, *symbolic* proof. The algorithmician (my contrived word to describe a person who is interested in design and analysis of algorithms), likes to see a *neat constructive* proof, that he can convert to an *efficient* algorithm.

The proofs of many great combinatorial theorems, particularly in graph theory, have this stigma that the proofs take recourse to case analysis. This situation itself seems to be unavoidable, when it comes to reckoning with collections of configurations made from finite sets. Another striking result, proved recently is the *strong perfect graph conjecture*, denoted SPGC. A graph $G = (V, E)$ is said to be *perfect* if the size of the maximum independent set (clique) is equal to the minimum number of cliques (independent sets) to cover for every induced subgraph of G. The SPGC states that a graph is *perfect* if and only it does not contain subgraphs homeomorphic to an odd cycle or an odd anti-cycle. This proof is not as cumbersome as the planarity proof and did not need the computer in its act. However, the proof is quite intricate and does involve case analysis arguments. This is typical of many theorems of graph theory which are based on what is known as a *forbidden subgraph* characterization.

I will not go into the details of this beautiful general theorem. I emphasize that a large number of interesting classes of perfect graphs arise in many practical applications. There exist polynomial time algorithms for many interesting graph theoretic optimization problems, such as colouring, covering, finding independence number, etc., on many subclasses of graphs, while these problems on general graphs are in general *hard* problems. In fact, this area is a fertile source of imaginatively solved problems for both mathematics and algorithmics.

Once again, I am fascinated by this dichotomy in metaphors. A theorem is beautiful, but its proof is not so elegant. However, the theorem and its implications lend themselves to polynomial time algorithms for many problems which are otherwise hard in general. Occasionally a theorem and its proof are both beautiful and elegant. Ironically, the proof may not be *effective* or the consequent algorithm may not be *efficient*.

A departure from this kind of bipolar situation has been the recent brilliant result of Manindra Aggarwal and his students at IIT, Kanpur, who showed that testing the primality of an integer can be carried out in deterministic polynomial time (see the authors' web page at IIT Kanpur and also[15]). This beautiful result uses a simple machinery of testing certain polynomial congruences. The work involved in checking these congruences is about the sixth power of the length of the integer to be tested. Of course the irony of algorithmic results is present in this situation too. There are certain probabilistic algorithms for testing primality of an integer which are used in practical applications, that are more efficient than this algorithm at the present.

The second class of questions in combinatorial mathematics is the *enumerative* kind. The questions are about the number of configurations. There are a whole lot of techniques based on combinatorial counting principles, generating functions, recurrence equations, Polya's theory of counting for inequivalent configurations etc. (see van Lint[16]). Suffice to say, that this body of knowledge is intimately connected with the science of algorithms in the area of analysis of algorithms. One is interested in the numbers of configurations to determine the average running time of an algorithm over all possible configurations.

Finally, the third class of questions is purely *constructive*. One is interested in *generating* the entire set or sub-sets of configurations. For example, collections of spanning trees, shortest paths, have further algebraic properties of theoretical and practical interest. The algorithmic computer scientist is interested in developing efficient algorithms to determine these configurations. It is here that the coming together of the two types of thinking, *mathematical* and *algorithmic* brings to a resounding success new ideas, thoughts, and results that enrich the beautiful corpus of results in both mathematics and computer science.

Many combinatorial proofs are also constructive in nature. They are indeed ingenious and sometimes surprisingly simple. I cite two particularly charming results from two different fields of mathematics, the first from graph theory and the second from number theory. (i) The necessary and sufficient condition for the maximality of a matching M in a bi-partite graph $G = (V_1, V_2, E)$ is the non-existence of any alternating path for M beginning and ending in unmatched vertices of M. (ii) The number $M_p = 2^p - 1$ is prime for a prime exponent p if and only if $v_{p-1} = 0$ in the sequence of integers $(v_1 = 4; \ldots v_i^2 = v_{i-1}^2 - 2; \ldots\}$, modulo M_p. The proofs (see van Lint[16] for the first and Knuth[10] for the second), of both these theorems are constructive in nature and both proofs lead to elegant, efficient polynomial time algorithms.

There are many combinatorial configurations which arise as *duals* of each other and often the counts of such configurations get related by what are known as extremal results. These results are typically of the form of maximum number of one type of configurations being equal to the minimum number of the other type of configurations. Such theorems are also known as *min-max* theorems. I indicate a few such theorems from the beautiful lore of these combinatorial gems. I refer the reader to (ref. 17) for a collection of beautiful results in combinatorics by the original authors and the recent book[18] on extremal combinatorics for more details. In fact the following theorems are also mutually related.

In a bi-partite graph (which we encountered above), the cardinality of a maximum matching is equal to the size of a minimum vertex cover. This theorem, named after *Konig*[19] and *Hall*[20] has many *avatars*. They are: the theorem of *Menger* that the maximum number of edge disjoint paths is equal to the minimum size of a vertex cut; of *Ford and Fulkerson* that the size of the maximum flow is equal to the size of the minimum cut, of *Konig*[19] on 0–1 matrices, that the maximum number of *disjoint* 1 elements in the matrix is equal to the minimum number of lines required to cover the 1 elements. The theorem of Dilworth[21], that in a partially ordered set, the maximum size of an antichain is equal to the minimum number of chains required to cover the poset, is also similar.

Coincidentally, there exist *elegant* polynomial time algorithms to determine all these combinatorial invariants. In fact almost all of these have constructive proofs that can be converted to algorithms. It is believed by researchers in this field, that combinatorial duality, min-max theorems and polynomial time solvability are intimately related from a philosophic point of view.

There is another metaphor in combinatorics known as a bijective proof. This typically arises in enumerative combinatorics where the equinumerous nature of two types (duals) of configurations is to be established. For example, many integer partition identities, lattice point counting paradigms, correspondences in tableau admit exquisite bijective proofs. These proofs have a close resemblance to the principles of design of efficient algorithms. Thus we see that mathematical and algorithmic ideas get entwined in many combinatorial contexts in resplendent ways.

Conclusions

Many applications draw heavily upon existing body of mathematical results and occasionally demand new mathematics. Contemporary computing science provides a new form of engendering new mathematical results. It provides new ways of looking at classical results.

I just mention a few exciting contemporary developments in both mathematics and computer science that have many connections based on deep mathematical and algorithmic thinking – Ramanujan's modular functions and expander graphs, computation of trillions of digits of constants like π; Ramanujan's asymptotic methods in combinatory analysis[22] and their implications in analysis of algorithms; the theory of elliptic curves (one of the three ingredients in the proof of Fermat's Last theorem by A. Wiles) and its role in modern public key cryptography[15,23]; new algebraic and number theoretic questions arising out of cryptography and coding theory[24] such as the use of number fields in integer factoring, divisors of algebraic curves in cryptosystems and codes[15]; lower bounds on computational complexity in algebraic models of computation; pseudo-randomness in algorithms and mathematics; many combinatorial optimization questions stemming out of computer algorithms for applications in the computer world of networks, circuit design; many new algorithmic questions in linear algebra with new applications.

Thus we live in an era when the two disciplines of mathematics and computer science have set up many strong interactions. These interactions and their interplay are leading to the enrichment of both disciplines. Together they may provide the right force multiplier effect to study and answer some of the deepest questions bothering mankind – of cognition, self, and thought.

1. Knuth, D. E., *The Art of Computer Programming Vol. 1 Fundamental Algorithms*, Addison Wesley, New York, 1973.
2. Dijkstra, E. W., *A Short Introduction to the Art of Programming*, Computer Society of India Press, Mumbai, 1977.
3. Appel, K. and Haken, W., Every planar map is four colourable. *Illinois J. Math.*, 1977, **21**, 429–567.
4. Manna, Z., *Logics of Programs*, Addison Wesley, 1987.
5. Knuth, D. E., Algorithmic thinking and mathematical thinking. *Am. Math. Monthly*, 1986, **81**, 322–343.
6. Edmonds, J., Paths, trees and flowers. *Can. J. Math.*, 1965, **17**, 449–467.
7. Garey, M. R. and Johnson, D. S., *Computers and Intractability – A Guide to the Theory of NP-completeness*, W. H. Freeman, San Francisco, 1979.
8. Lemmermeyer, F., *Reciprocity Laws: From Euler to Eisenstein*, Springer, New York, 2000.
9. Edwards, H. M., *Fermat's Last Theorem*, Springer, New York, 1977.
10. Knuth, D. E., *The Art of Computer Programming Vol. 2 Seminumerical algorithms*, Addison Wesley, New York, 1981.
11. Yan, S. Y., *Number Theory for Computing*, Springer, New York, 2000.
12. Knuth, D. E., *The Art of Computer Programming Vol. 3 Sorting and Searching*, Addison Wesley, New York, 1981.
13. Aho, V., Hopcroft, R. E. and Ullman, J. D., *Design and Analysis of Algorithms*, Addison Wesley, New York, 1982.
14. Aigner, M. and Ziegler, G. M., *Proofs from The Book*, Springer, New York, 1998.
15. Abhijit Das and Veni Madhavan, C. E., *Public Key Cryptography: Theory and Practice*, Manuscript of a forthcoming book, 2005.
16. van Lint, J. H. and Wilson, R. M., *A Course in Combinatorics*, Cambridge University Press, London, 1992.
17. Gessel, I. and Gian-Carlo Rota (eds), *Classic Papers in Combinatorics*, Birkhauser, 1987.
18. Jukna, S., *Extremal Combinatorics with Applications to Computer Science*, Springer, New York, 2001.
19. Konig, D., *Math. Ann.*, 1916, **77**, 453–465.
20. Hall, P., On representatives of subsets. *J. London Math. Soc.*, 1935, **10**, 26–30.
21. Dilworth, R. P., A decomposition theorem for partially ordered sets. *Ann. Math.*, 1950, **51**, 161–166.
22. Hardy, G. H. and Ramanujan, S., Asymptotic formulae in combinatory analysis. *Proc. London Math. Soc.*, 1918, **17**, 75–115.
23. Koblitz, N., *A Course in Number Theory and Cryptography*, Springer, New York, 1996.
24. MacWilliam, F. J. and Sloane, N. J. A., *The Theory of Error Correcting Codes*, North Holland, New York, 1977.